Vorlesung
P2P Netzwerke
10: Multicast

Dr. Dominic Battré
Complex and Distributed IT-Systems
dominic.battre@tu-berlin.de

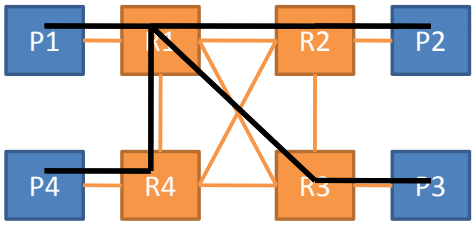
Inhalt

- Application-Layer Multicast
- Scribe
- SplitStream
- BitTorrent

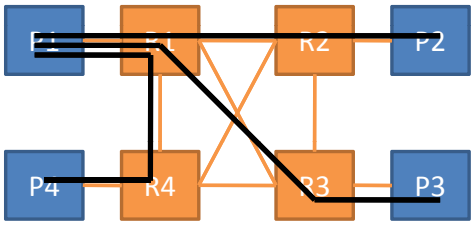
Multicast

- IP Multicast ist eine tolle Idee, aber
  - kaum unterstützt
  - daher nicht praktisch einsetzbar
- Alternativen?
- Application-Layer Multicast!

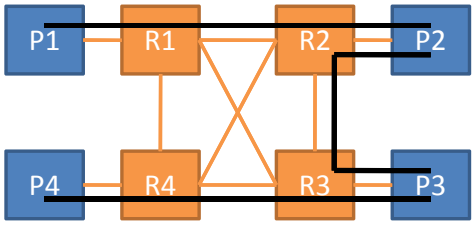
Application-Layer Multicast



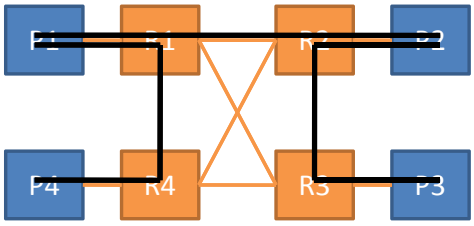
Max Stress = 1
Avg Stretch = 1



Max Stress = 3
Avg Stretch = 1



Max Stress = 1
Avg Stretch = 2



Max Stress = 2
Avg Stretch = 1,33

Scribe: Literatur

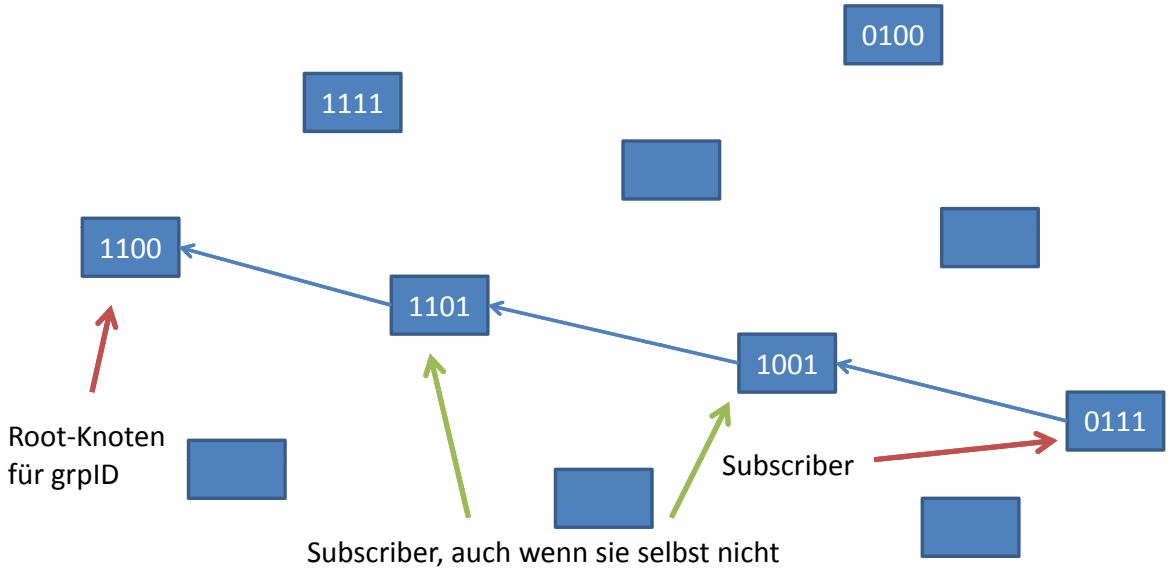
[1] M. Castro, P. Druschel, A-M. Kermarrec, A. Rowstron: "SCRIBE: A large-scale and decentralised application-level multicast infrastructure", IEEE Journal on Selected Areas in Communications (JSAC) (Special issue on Network Support for Multicast Communications), 2002.

[2] M. Castro, P. Druschel, A-M. Kermarrec, A. Nandi, A. Rowstron, A. Singh: "SplitStream: High-bandwidth multicast in a cooperative environment", OSP'03,Lake Bolton, New York, October, 2003.

Scribe: Basis

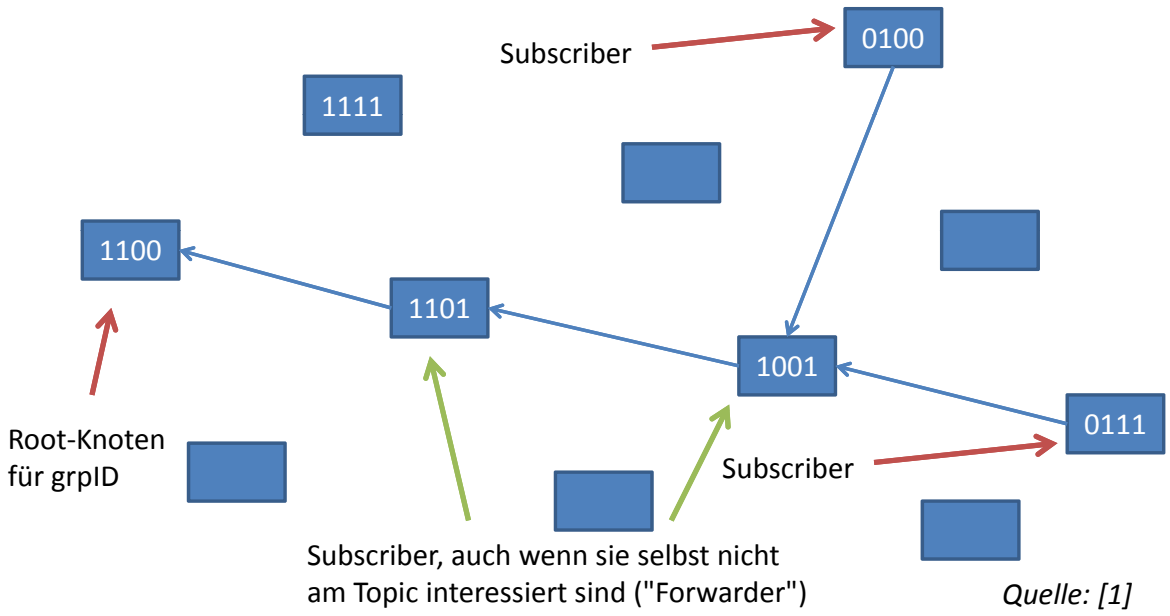
- Baut auf Pastry auf
- Multicast-Kommunikation in Gruppen
- Jede Gruppe hat eine ID grpID
- Der Root-Knoten für grpID in Pastry verwaltet die Gruppe
- Join von Knoten n:
  - Sende Nachricht JOIN an grpID
  - Jeder Knoten auf dem Weg überprüft, ob er in der Gruppe ist
  - Falls ja: Füge n zur Kinder-Liste für grpID hinzu
  - Nachricht wird nur weitergeleitet, wenn der Knoten noch nicht in der Gruppe war
- Es wird dadurch ein Baum für jede Gruppe aufgebaut

Scribe und SplitStream



Quelle: [1]

Scribe und SplitStream



Quelle: [1]

Scribe: Verlassen einer Gruppe

- Leave von Knoten n aus Gruppe grpID:
  - Wenn Kinder-Liste leer ist, route Leave-Message Richtung Root-Knoten
  - Sonst Markiere Gruppe nur als "verlassen"
- Wenn eine Leave-Nachricht von einem Kind empfangen wurde:
  - Lösche Kind aus Kinder-Liste
  - Wenn Kinder-Liste leer ist, und der Knoten selbst nicht in der Gruppe ist:
    - Sende Leave-Nachricht an Parent und lösche die Gruppe lokal

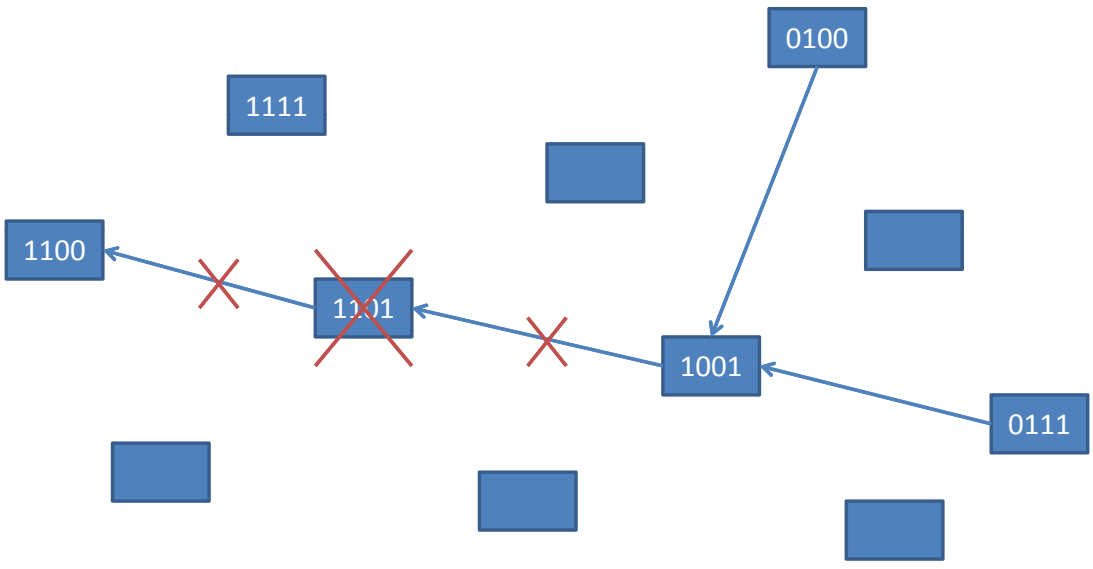
Scribe: Senden einer Nachricht

- Kanonische Lösung:
  - Nachrichten werden immer zur Wurzel geschickt
  - Dazu wird das NodeHandle gecacht
  - Wenn es falsch oder unbekannt ist, wird per Pastry-Routing die Wurzel gesucht
  - Von der Wurzel aus sendet jeder Knoten die Nachricht an seine Kind-Knoten
- Alternative:
  - Verteilen der Nachricht direkt vom aktuellen Knoten aus
  - Jede Nachricht beinhaltet eine Liste von Knoten, die die Nachricht bereits gesehen haben
  - Die Nachricht wird an den Parent-Knoten und an alle Kind-Knoten verschickt

Scribe: Reparatur

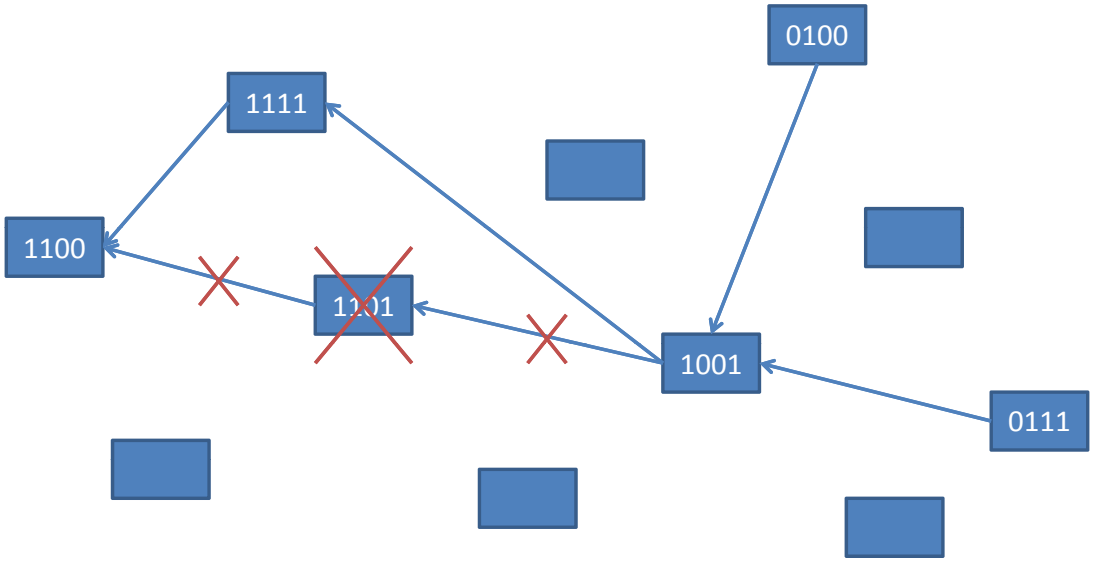
- Die Wurzel sendet regelmäßig "Heartbeat"-Nachrichten
- Normale Nachrichten sind implizit Heartbeats
- Wenn der Heartbeat vom Parent-Knoten ausbleibt, sendet er eine neue JOIN-Nachricht
- Dadurch wird der Baum repariert
- Die Wurzel repliziert ihren Zustand (wie gewohnt) über das Leafset
- Ein neuer Root-Knoten übernimmt direkt nach dem Ausfall der Wurzel das Verteilen und Aussenden der Heartbeats

Scribe



Quelle: [1]

Scribe

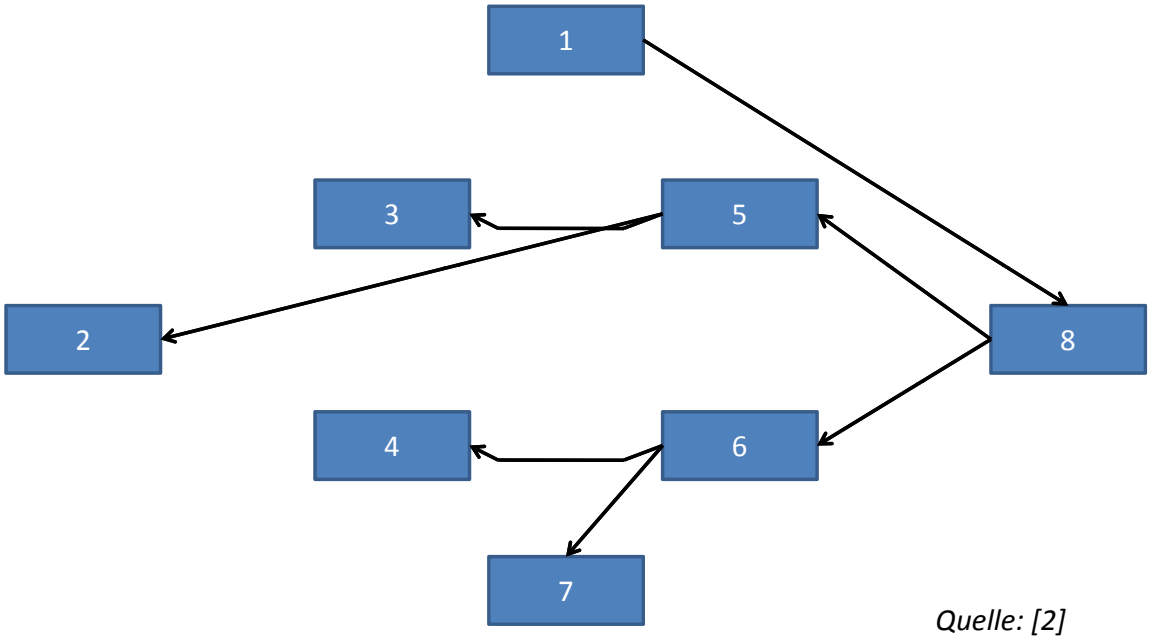


Quelle: [1]

Scribe: Probleme

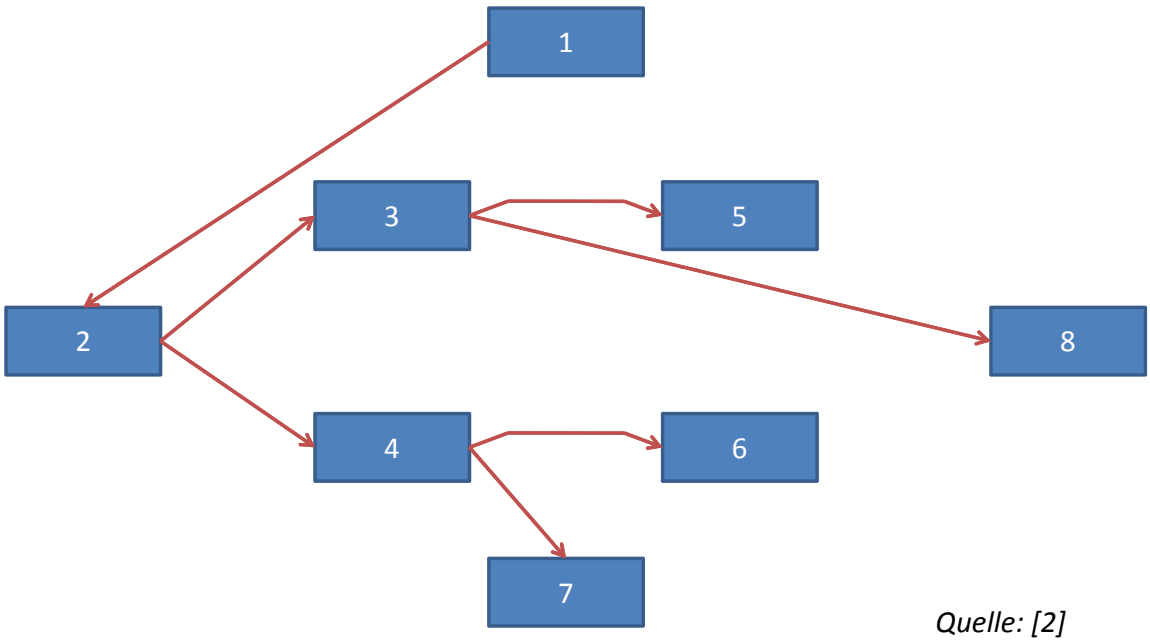
- Innere Knoten des Baumes müssen weiterleiten
- Blattknoten müssen nur empfangen
- Für Baum mit Fanout f, i inneren Knoten, b Blättern gilt:
  - $(f-1) \cdot i + 1 = b$
  - ➔Anteil an inneren Knoten ca. 1/f
- Last der inneren Knoten:
  - Ausgangs-Bandbreite f-mal der Datenstrom
- ➔ Sehr ungleiche Lastverteilung im Baum
- Lösung: **SplitStream**
- Grundidee:
  - Aufteilen des Datenstroms in mehrere Teilströme
  - Für jeden Teilstrom einen eigenen Baum aufbauen

SplitStream



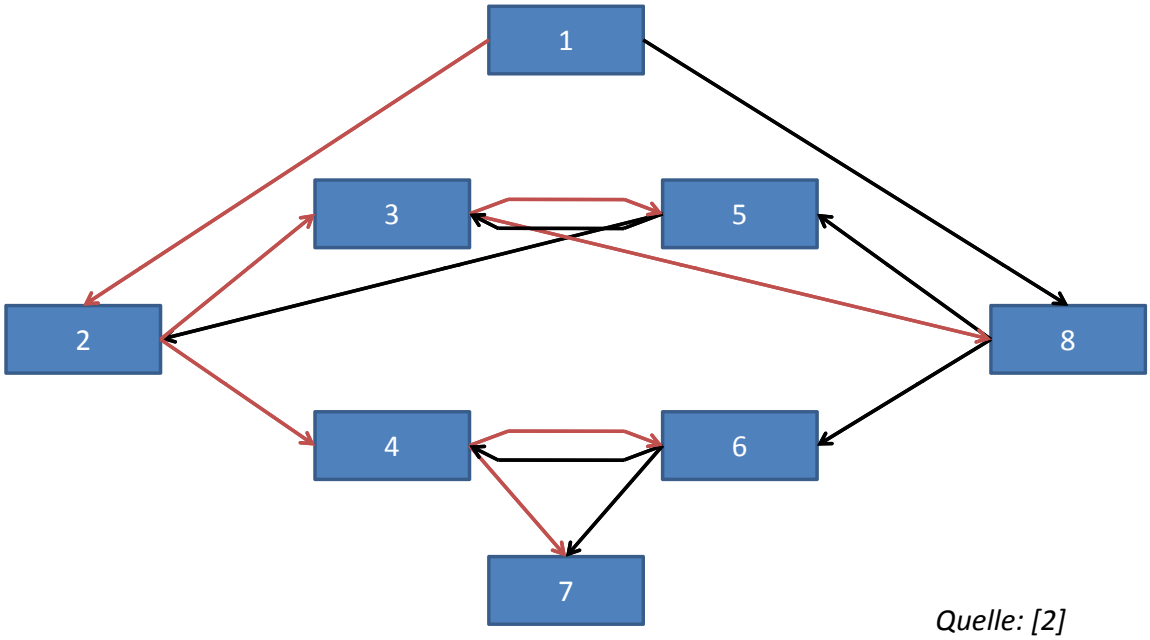
Quelle: [2]

SplitStream



Quelle: [2]

SplitStream



Quelle: [2]

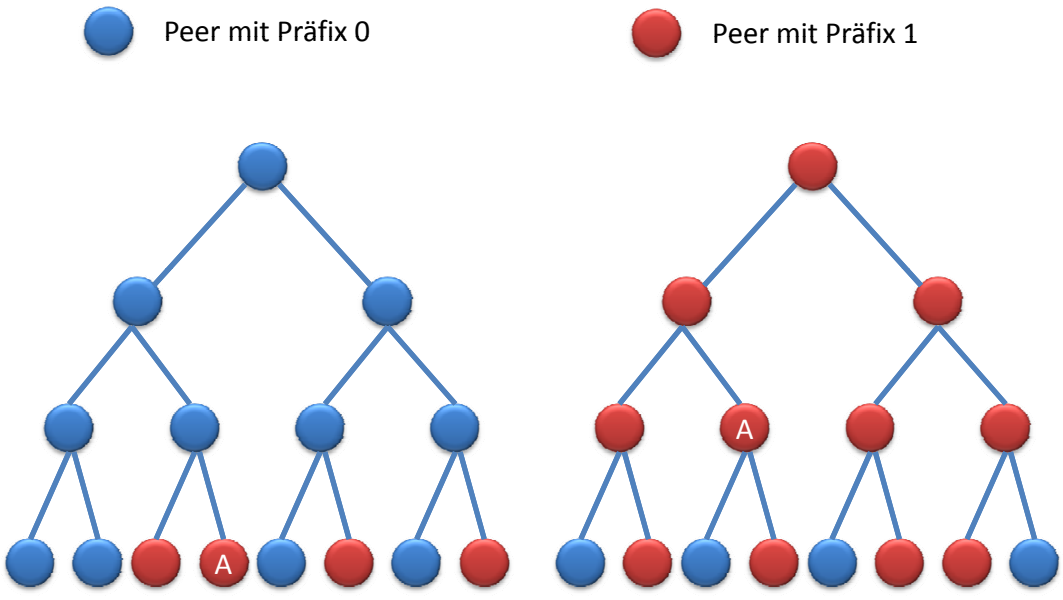
SplitStream

- Datenstrom mit Bandbreite B wird verteilt
- Der Datenstrom wird in k Stripes eingeteilt
- Für jeden Stripe wird ein eigener Baum aufgebaut
- Es muss nicht jeder Knoten alle Stripes empfangen
  - Warum macht das Sinn?
- Jeder Knoten kann seine Upload- und Download-Bandbreite in Schritten von B/k kontrollieren
- Dadurch Berücksichtigung von:
  - Knoten mit unterschiedlicher Bandbreite
  - Ungleiche Up- und Download-Bandbreite

Aufbau der Bäume

- Jeder Stripe bekommt eigene ID
- Die ID's unterscheiden sich in der höchsten Ziffer
- Jeder Empfänger-Knoten meldet sich bei allen Stripe-IDs an
- Zu jeder ID wird ein eigener Scribe-Baum aufgebaut
- Je dichter ein Knoten an der Wurzel ist, desto länger ist der gemeinsame Präfix
- Ideal für diese Konstruktion ist: $k = 2^b$
- Jede Route von einem Knoten zu einer Stripe-ID stellt im ersten Schritt Übereinstimmung in der ersten Stelle sicher
- Die weitere Route verläuft nur über Knoten mit der gleichen ersten Stelle
- Alle Bäume haben also disjunkte Mengen innerer Knoten

Aufbau der Bäume



Kontrolle der Bandbreite

- Die eingehende Bandbreite entspricht der Menge der abonnierten Stripes, oder
- max. 1, wenn kein Stripe abonniert wurde, aber der Knoten innerer Knoten eines Stripe-Baumes ist
- Aber: die ausgehende Bandbreite könnte überschritten werden
- Methode in *Scribe* zur Kontrolle:
  - Wenn ein Knoten einen neuen Kind-Knoten bekommt, der die Bandbreite überschreitet:
  - Sende dem Knoten bisherige Kind-Liste
  - Der Knoten versucht sich bei dem Kind mit der geringsten Latenz anzumelden
  - Dies wird rekursiv fortgesetzt
- Diese Methode führt in SplitStream zu Problemen! (Blatt Knoten kann evtl. kein Kind annehmen, weil er in anderem Baum innerer Knoten und damit ausgelastet ist)

Kontrolle der Bandbreite

- Jeder Knoten nimmt jedes neue Kind zunächst an
- Wenn seine Bandbreite überschritten wird:
  - Selektiere das Kind mit dem kürzesten gemeinsamen Präfix
  - Hänge diese Kind ab
- Das verwaiste Kind muss sich einen neuen Parent-Knoten suchen
  - Zunächst rekursiver Versuch, sich bei den anderen Kindern des alten Parent-Knoten einzuhängen
  - Dabei werden nur Kinder mit zum Stripe passenden Präfix genutzt
  - Sonst nutze die "Spare capacity group"
- Spare Capacity Group
  - Scribe-Gruppe in der alle Knoten mit freien Kapazitäten sind

Wie verwende ich SplitStream?

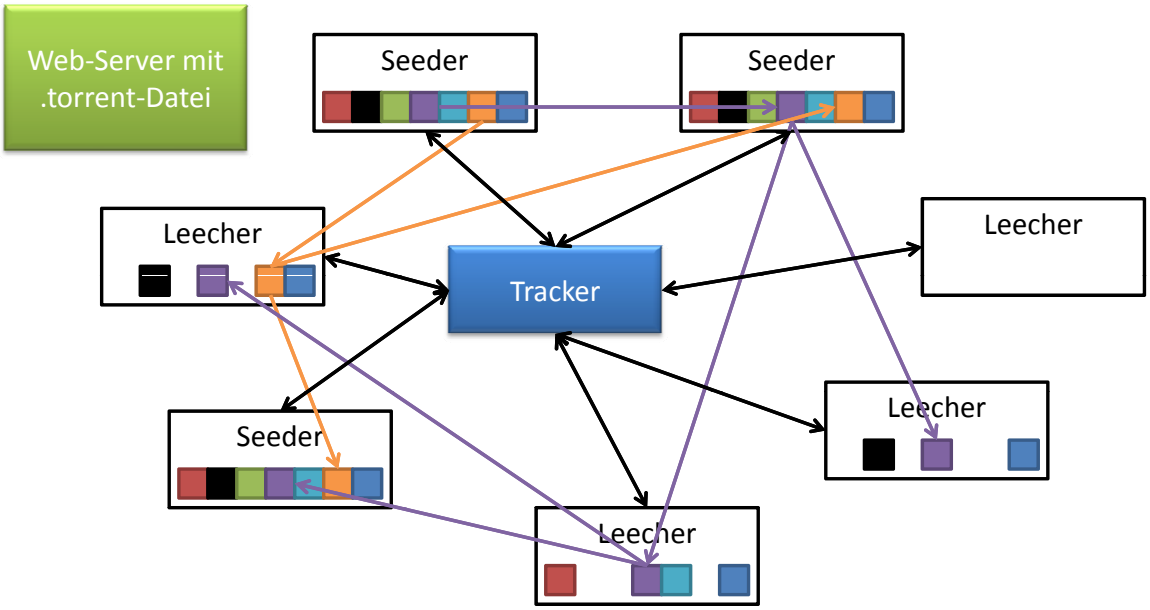
- Jeder einzelne Stripe könnte "Aussetzer" haben:
- Reparaturzeiten nach Knotenausfall, Umorganisation, etc.
- Daher: Einsatz von fehlertoleranter Kodierung
  - Empfang aller k Stripes ist nicht nötig
  - Ausfall einzelner Stripes kann kompensiert werden

BitTorrent: Literatur

[1] Bram Cohen: "Incentives Build Robustness in BitTorrent", 2003.

[2] Ashwin R. Bharambe, Cormac Herley, Venkata N. Padmanabhan: "Analyzing and Improving BitTorrent Performance", Microsoft Research Technical Report MSR-TR-2005-03, 2005.

BitTorrent



BitTorrent: Funktionsweise

- Ein Peer lädt eine Torrent-Datei von einem Webserver
  - Dateiinfos: Name, Länge etc.
  - URL des Trackers
  - SHA1-Werte der Dateiblöcke
- Es muss mind. einen Seeder anfänglich geben
- Ein Peer meldet sich beim Tracker an
- Der Tracker schickt eine zufällige Auswahl an Peers (ca. 40)
- Der Peer baut Direktverbindungen zu diesen Peers auf
- Der Peer lädt jetzt die Datei blockweise von seinen Nachbar-Peers runter
- Wenn ein Peer weniger als 20 Nachbarn hat, holt er neue vom Tracker

BitTorrent: Details

- Versendung der Blöcke über TCP-Verbindungen
- Jeder Block wird in Sub-Blöcke aufgeteilt (16 KB)
- Diese werden im Pipelining-Verfahren verschickt
- Bevor ein neuer Block angefordert wird, werden erst fehlende Sub-Blöcke von alten Blöcken angefordert
- Welches Paket wird als nächstes angefordert?
  - Zufällige Auswahl genügt nicht
  - → Coupon Collector Problem

Coupon Collector's Problem

- Datei besteht aus m Teilen
- Jeder Peer hat genau einen zufällig ausgewählten Teil
  - → Es müssen Ω(m log m) Peers teilnehmen, damit mit konstanter Wahrscheinlichkeit jeder Teil verfügbar ist
- Verteilt man m Teile auf m Peers,
  - haben 1/e der Peers gar keine Teile
  - einige Peers O(log m) Teile
- → Ungleichgewicht

BitTorrent: Details

- Welches Paket wird als nächstes angefordert?
  - "Random First Piece": Erstes Paket wird zufällig gewählt
    - Um schnell an Daten zum Weiterverteilen zu kommen
    - Sobald erstes Paket vollständig, wechsel zu Rarest First
  - "Rarest First": Paket, welches am wenigsten repliziert ist unter den bekannten Peers
  - "Endgame Mode": Am Ende alles von überall anfordern

BitTorrent: Tit for Tat

- Wie verhindere ich, dass jemand nicht weiterverteilt?
- Grundsätzliche Idee:
  - Ich schicke nur denen etwas, die auch selbst etwas weiterverteilen
- Probleme:
  - Peers, die noch nichts zum Verteilen haben
  - Peers, die eine schlechte Anbindung haben, und von denen daher niemand etwas runterladen möchte
  - Woher bekomme ich die Informationen?

BitTorrent: Choking

- Eine Verbindung kann aktiv oder passiv sein (choking, keine Anfragen werden beantwortet)
- Es sind immer nur 4 Verbindungen aktiv (unchoking)
- 3 werden über die besten Download-Raten bestimmt
  - 20 sec. rollierender Durchschnitt
- Alle 10 sec. werden die aktiven Verbindungen neu gewählt
  - wegen TCP ramp up
- Zusätzlich: Ein "Optimistic Unchoke"
  - Eine zufällige Verbindung wird aktiviert
  - Für neue Peers und solche mit schlechter Upload-Bandbreite
  - Rotiert alle 30 sec.
- Wenn keine Pakete mehr ankommen
  - Mehr als ein Optimistic Unchoke
- Wenn Download komplett
  - Wähle aktive Peers mit höchster erzielbarer Upload-Rate

Zusammenfassung

- Problemstellung: Multicast, d.h. Verteilung von Daten an eine große Empfängergruppe
- IP-Level Multicast wäre optimal
- hat sich aber nicht durchgesetzt
- Daher: Application-Level Multicast
- Mit P2P gut zu realisieren
- Prinzip: Baumförmige Struktur aufbauen
- Probleme: Fairness sowie gute Ausnutzung der vorhandenen Kapazitäten
- Lösungen: Daten in mehrere Teile zerhacken, für jeden Teil eigenen Baum aufbauen
- Systeme: Scribe, SplitStream, BitTorrent